Feature-Oriented Programming with Family Polymorphism

Fuminobu Takeyama
Shigeru Chiba

Tokyo Institute of Technology

Chiba Shigeru Group
Tokyo Tech

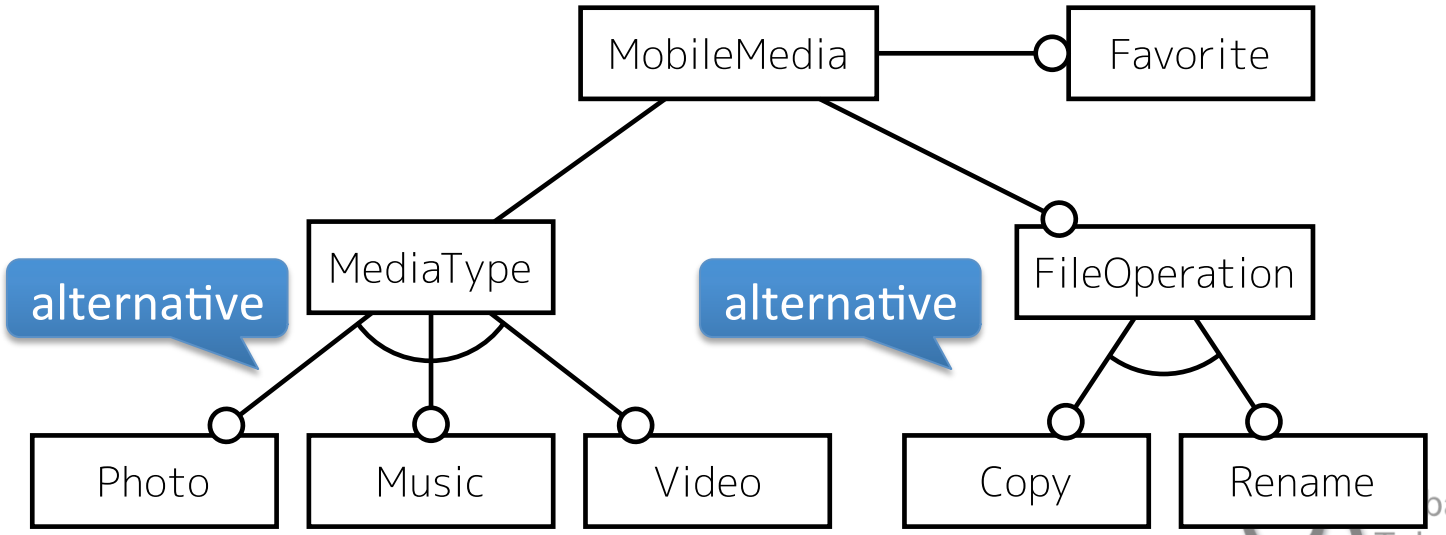
Feature-oriented programming (FOP)

- modularizes code snippets related to the same feature
  - useful for software product lines (SPL)
- A feature = aspects + classes
  - To improve composability
    - *i.e.*, easy to add/remove code snippets related to a feature
  - In AHEAD, a typical FOP language, refinement + classes
- Selecting a feature
  - Aspects and classes related the feature is compiled

Chiba Shigeru Group
Tokyo Tech

MobileMedia SPL [T. Young, et al., AOSD 2005 demo]

- Multimedia viewer for mobile devices
- Alternative feature
  - **original:** select 1 feature from alternative features
  - select a subset of features from alternative features



Code clones among features

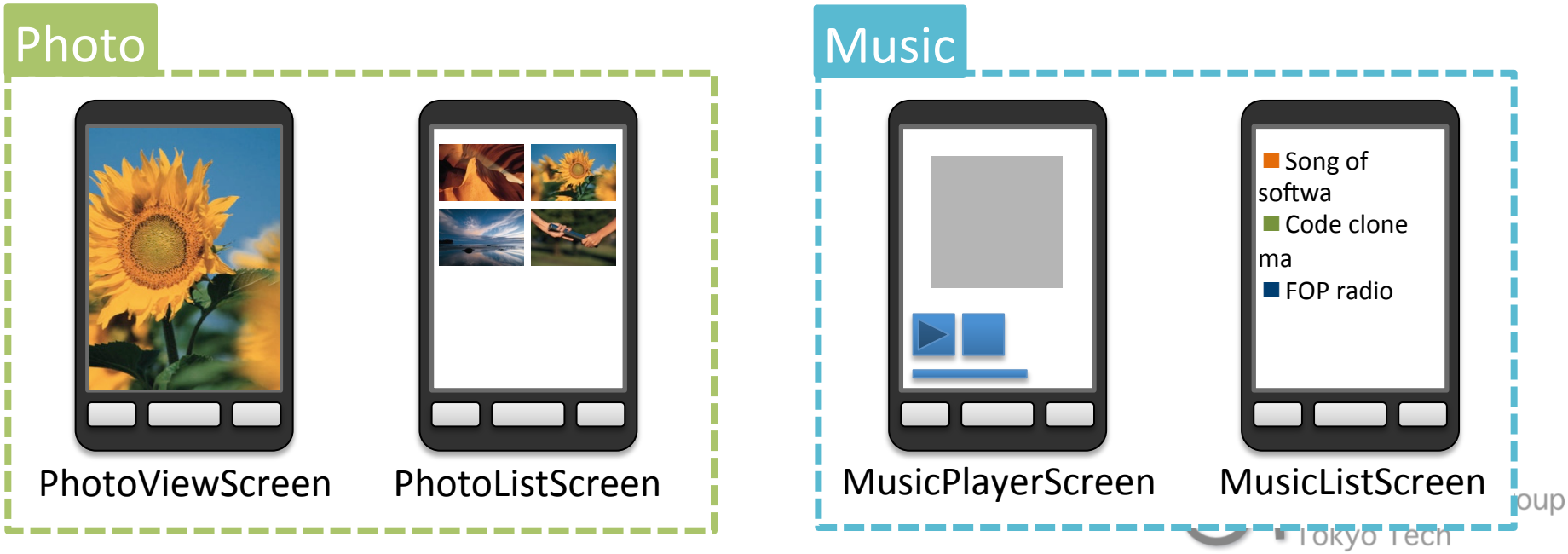
[S. Schulze, GPCE 2010]

- Code clones are found especially in
  - alternative features
  - derivatives
- These clones cannot be removed easily
  - Not only classes but also aspects
  - Class names in those clones are different

Chiba Shigeru Group
Tokyo Tech

Clones among alternative features 1/2

- Because alternative feature have similar behavior and structures of classes
  - Both Photo and Music have a list of media
    - show/play a selected medium



Clones among alternative features 2/2

Photo

```
reviser PhotoTypeInitializer
              extends Application {
  private PhotoListScreen screen;
  private PhotoController cont;
  void startApp() {
    screen = new PhotoListScreen();
    cont = new PhotoController();
    super.startApp();
}}
```

```
class PhotoController ext… Cont… {
  boolean handleCommand(Command c) {
    if (c == OPEN) {
      String sel = getSelected();
      Display.setCurrent(
        new PhotoViewScreen(sel));
    } else if (...) { ... }
}}
```

```
class PhotoViewScreen ext… Screen {…}
```

Music

```
reviser MusicTypeInitializer
              extends Application {
  private MusicListScreen screen;
  private MusicController cont;
  void startApp() {
    screen = new MusicListScreen();
    cont = new MusicController();
    per.startApp();
```

```
class MusicController ext… Cont… {
  boolean handleCommand(Command c) {
    if (c == OPEN) {
      String sel = getSelected();
      Display.setCurrent(
        new MusicPlayerScreen(sel));
    } else if (...) { ... }
```

```
class PhotoViewScreen ext… Screen {…}
```

Clones in aspects

Class names are different

Code clones among derivatives 1/2

- A derivative
  - a special feature needed only when a certain set of features are selected
- PhotoCopy derivative
  - needed only when Photo and Copy are selected
  - Adding “Copy” to PhotoListScreen

Code clones among derivatives 2/2

- Different derivative for every combination
  - The number of derivative might explode

PhotoCopy

```
reviser AddCopyToPhoto extends PhotoListScreen {
  void initMenu() {
    addCommand(new CopyCommand());
}}
```

MusicRename

```
reviser AddRenameToMusic extends MusicListScreen {
  void initMenu() {
    addCommand(new RenameCommand());
}}
```

PhotoRename

MusicCopy

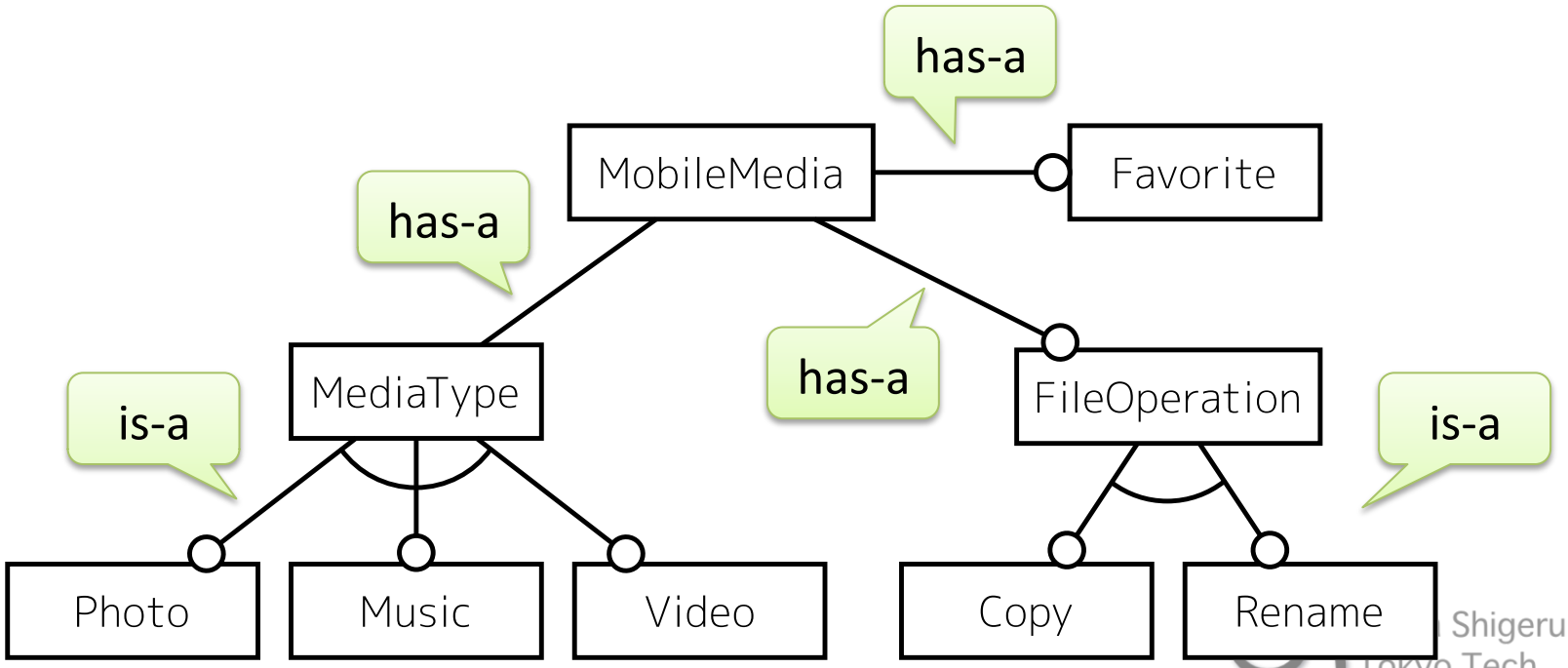
VideoCopy

VideoRename

The cause of these clones

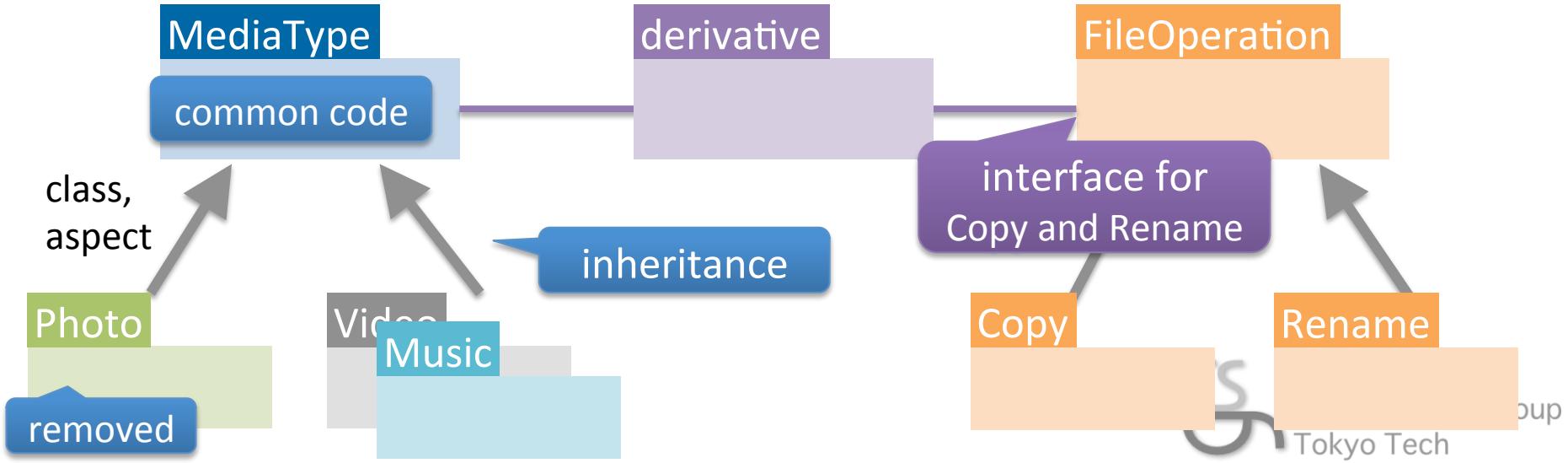
A topic for discussion

- Our observation is there are two types of edges
  - It is said that an edge represents a *has-a* relation
  - Some edges normally represent *is-a*



Avoiding code clones with inheritance

- A feature can be implemented by extending another
  - Common codes can be removed into super feature
- A generic derivative
  - A super feature is used as interface for its sub features



Family polymorphism with aspects

- A family is regarded as a feature
  - consist of aspects and virtual classes
- A virtual class
  - can be overridden by classes of sub feature
    - The real class referred by a virtual class is depends on feature
  - reduce difference of class names
    - in classes and aspects

Chiba Shigeru Group
Tokyo Tech

Feature modules in FeatureGluonJ

- Feature
  - classes belong to one feature
    - declare at the top of source code

Photo

```
feature Photo {}
```

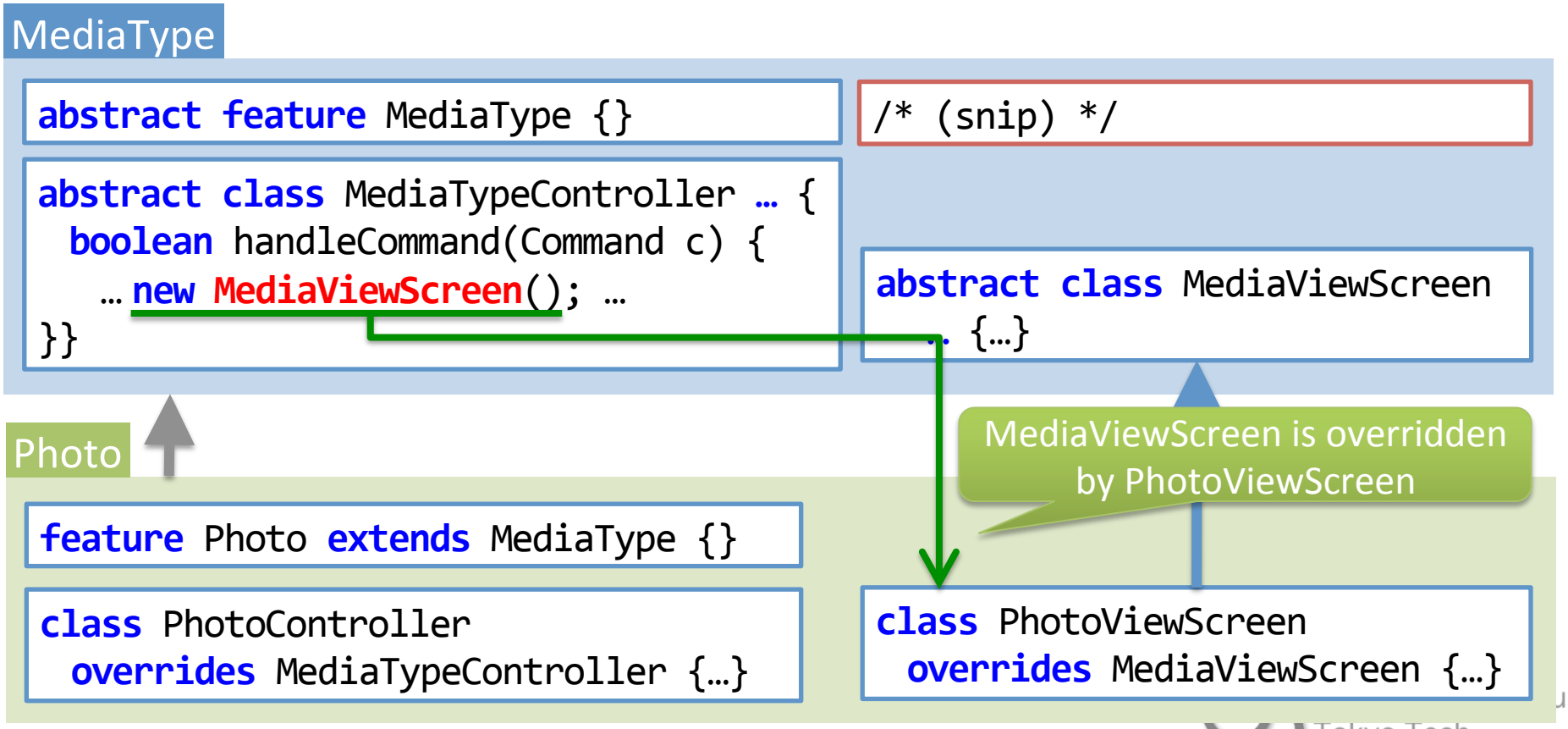
```
feature Photo;
class PhotoController … {
  boolean handleCommand(Command c) {
    /* (snip) */
}}
```

```
feature Photo;
class PhotoViewScreen … {…}
```

```
feature Photo;
reviser PhotoTypeInitializer … {
  /* (snip) */
  void startApp() {
    /* (snip) */
}}
```

PhotoFeature implemented with inheritance

- Use **overrides**, not **extends**
  - X overrides Y: X is replaced with Y that extends original X



Family polymorphism with revisers 1/2

- Reviser in GluonJ [S. Chiba, et al, OOPSLA 2010]
  - around advices with an execution pointcut
  - intertype declarations
  - can belong to a feature

```
reviser MediaTypeInitializer extends Application {
  …
  private MediaController cont;
  …
  void startApp() {
    cont = new MediaController();
    …
    super.startApp();
}}
```

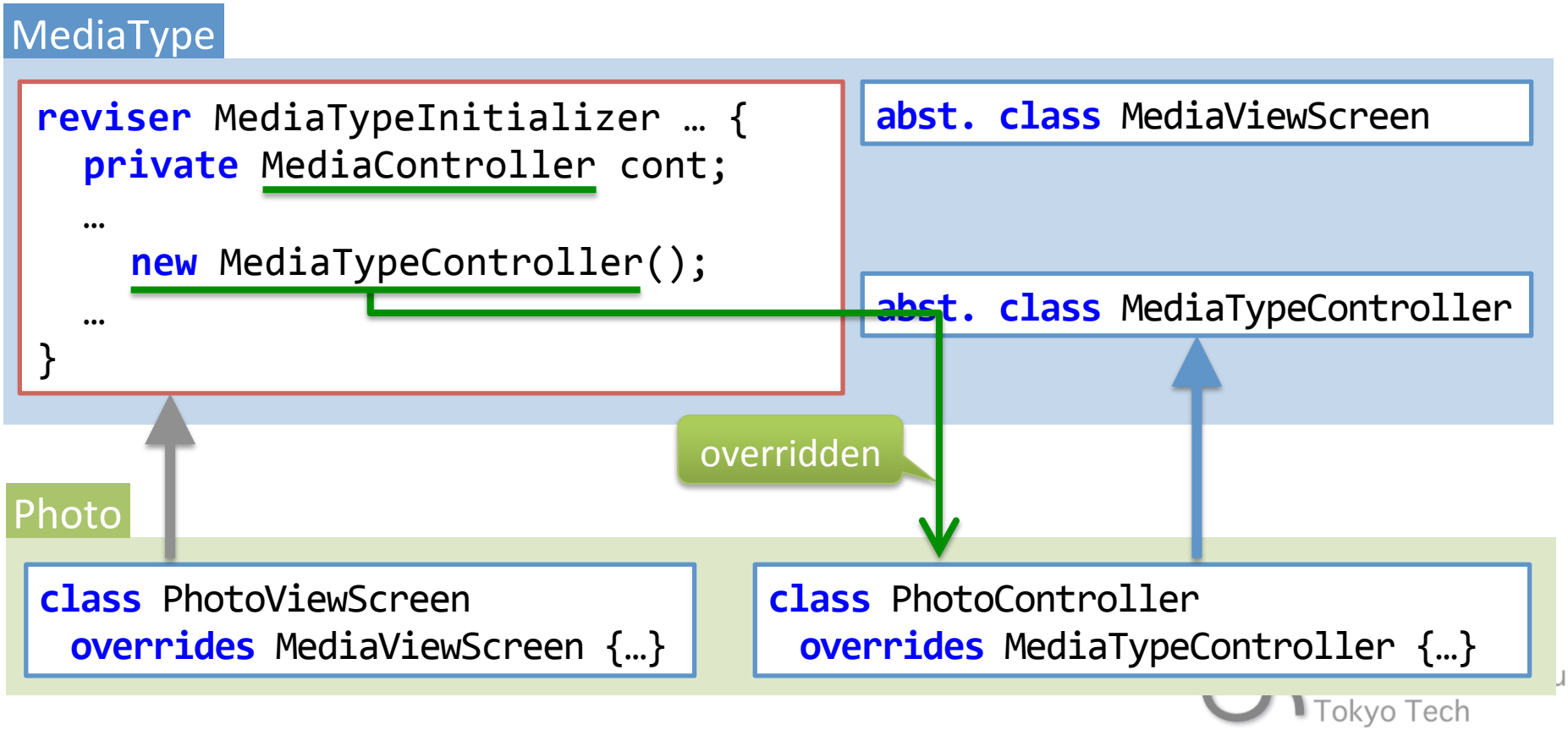
Intertype declaration

Advice

```
class Application {
  void startApp() {
    // executed when this application starts
}}
```

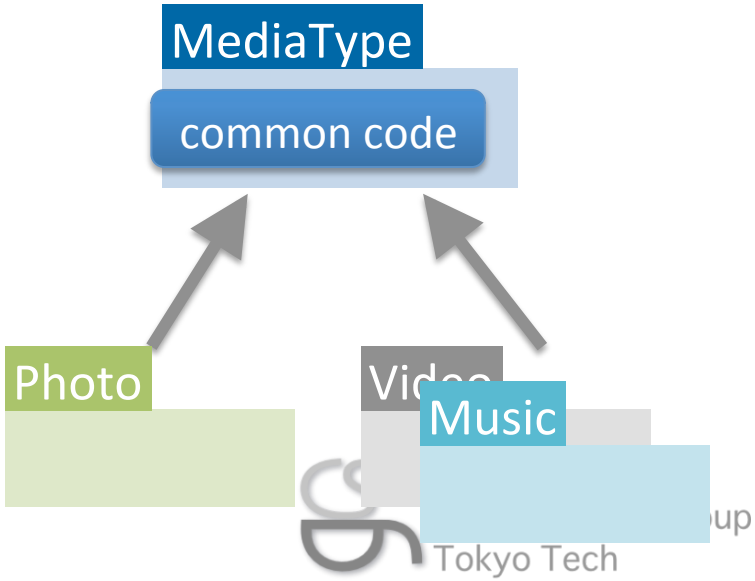
Family polymorphism with revisers 2/2

- No need to define the revisers for each feature
  - Virtual classes in a reviser are overridden as well

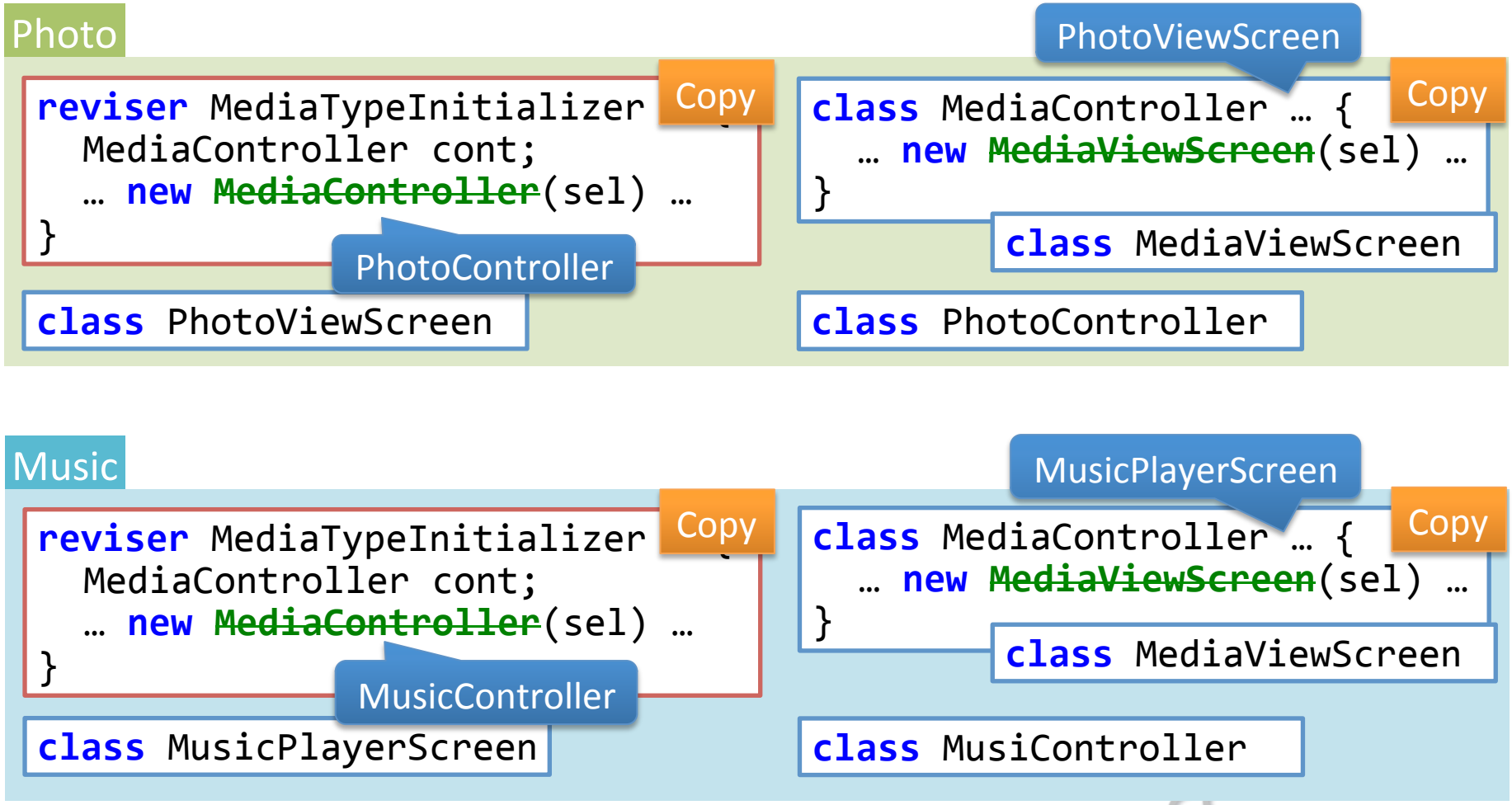


Copy semantics 1/2

- Classes and revisers of super feature are copied to sub-features
  - visible only in the features
- Selecting multiple feature from sub features
  - multiple copies of a reviser will be executed.



Copy semantics 2/2



Chiba Shigeru Group
Tokyo Tech

Implementation of naive derivatives

- Derivatives represents connection between features
- 2 steps to access classes belonging other features
  - **`import feature`**
  - Feature qualified class access, `::`

Derivative clones are still remaining

PhotoCopy

```
feature PhotoCopy {
  import feature p: Photo;
  import feature c: Copy;
}
```

alias of imported feature

```
reviser AddCopyToPhoto extends p::PhotoListScreen {
  void initMenu() {
    addCommand(new c::CopyCommand());
}}
```

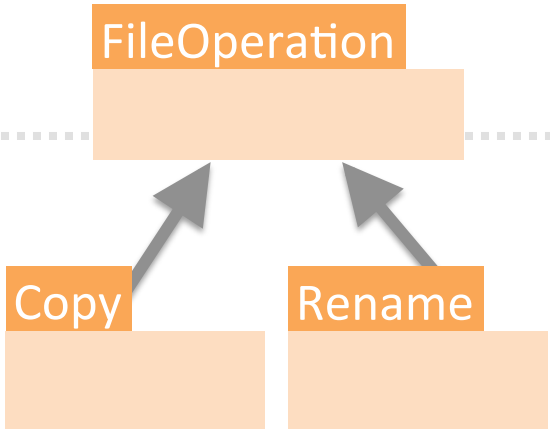
Copy's CopyCommand

PhotoRename

MusicCopy

VideoRename

A generic derivative



- used to implement other derivatives
  - A super feature is used as interface
    - What virtual classes the feature provides?
  - `import feature` will be overridden by sub-derivative

MediaTypeFileOp

```
feature MediaTypeFileOp {
  abstract import feature mt: MediaType;
  abstract import feature fo: FileOperation;
}
```

```
reviser AddCmdToMediaList extends mt::MediaListScreen {
  void initMenu() {
    addCommand(new fo::FileOpCommand());
}}
```

mt have a class overriding MediaListScreen

Derivatives extending the generic derivative

- Code clones are removed
  - Redundant feature definitions are still remaining

MediaTypeFileOp

```
feature MediaTypeFileOp {
   abstract import feature mt: MediaType;
   abstract import feature fo: FileOperation;
}
```

```
reviser AddCmdToMediaList extends mt::MediaListScreen {
   void initMenu() {
      addCommand(new fo::FileOpCommand());
}}
```

PhotoCopy

```
feature PhotoCopy extends MediaTypeFileOp {
   import feature mt: Photo;
   import feature fo: Copy;
}
```

Overrides alias with the same name

PhotoRename

```
feature PhotoRename extends MediaTypeFileOp {
   import feature mt: Photo;
   import feature fo: Rename;
}
```

A reviser is derived

Auto generation of the derivatives

- derivatives extending the generic one
  - no need to create new derivatives when a new feature is added

MediaTypeFileOp

```
feature MediaTypeFileOp defines forevery(mt, fo) {
  abstract import feature mt: MediaType;
  abstract import feature fo: FileOperation;
}
```

```
reviser AddCmdToMediaList extends mt::MediaListScreen {
  void initMenu() {
    addCommand(new fo::FileOpCommand());
}}
```

Chiba Shigeru Group
Tokyo Tech

Related work 1/2

[C. Saito, et al., J. of functional prog. (18) 2008]

- Lightweight family polymorphism
  - Similar semantics from the view point of virtual class
    - Virtual classes (and revisers) can be described only in top level
    - No subtyping between classes from different family

- CaesarJ

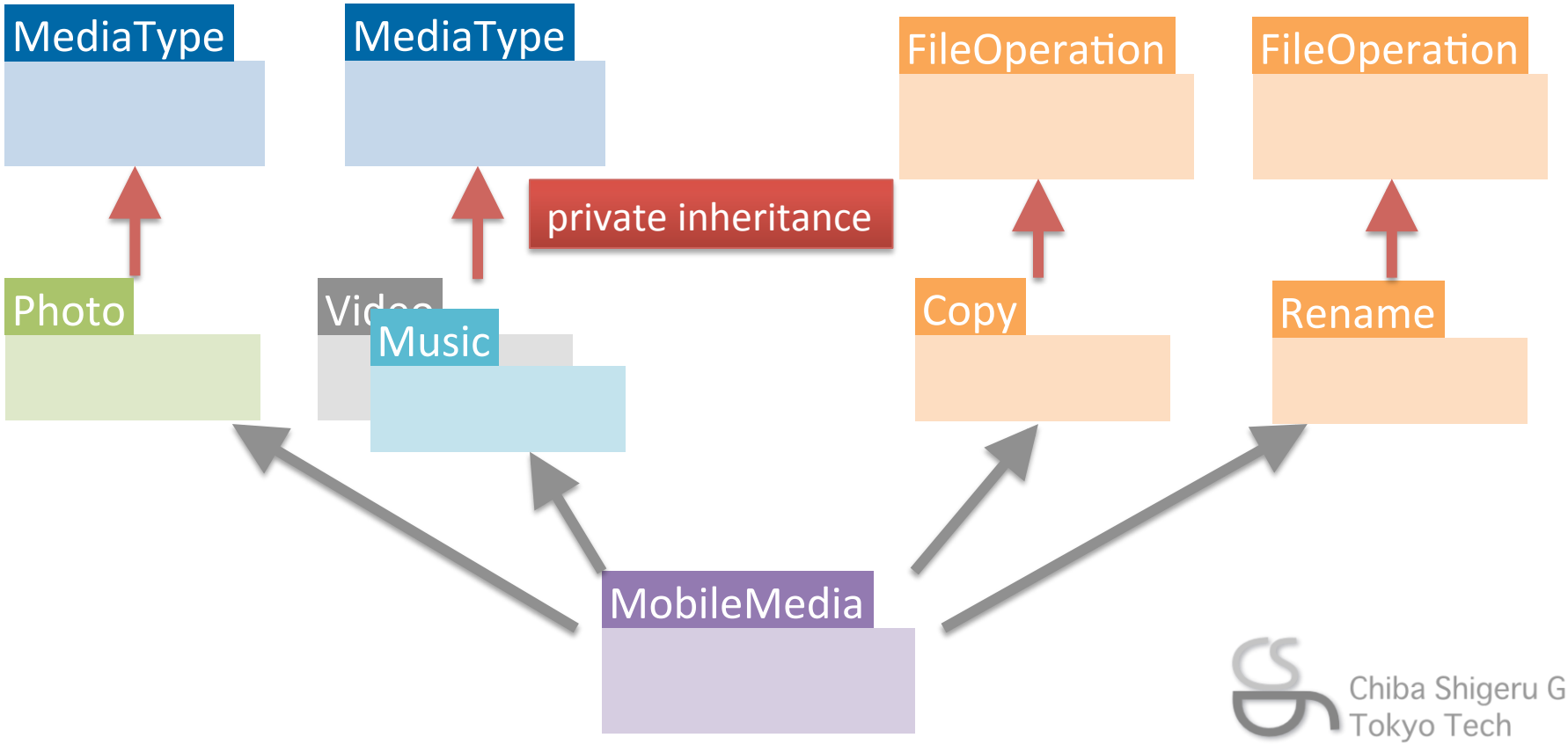
[I. Aracic , et al., TAOSD I 2006]

  - language supporting virtual classes and aspects
  - Aspects in CaesarJ are limited
    - No intertype declarations
    - (?) cannot handle join points in other feature (cclass)
      - Does anyone know the detail of CaesarJ's semantics?

Chiba Shigeru Group
Tokyo Tech

Related wrok 2/2

- FOP only with virtual classes [V. Gasiunas, et al., AO Product Line Eng.]
  - needs private inheritance to prevent the code clones



Chiba Shigeru Group
Tokyo Tech

Conclusions

- FeatureGluonJ
  - family polymorphism / virtual classes for FOP
    - supports aspects
  - removes code clones
    - among alternative features
    - among derivatives
- Future work
  - Formal definition of type system
  - Other types of clones
    - code for resolving conflict of revisers

Chiba Shigeru Group
Tokyo Tech